Implementation Document

for

Automated Lecture Hall Booking Portal

Version 1.0

Prepared by

Group 12 **Group Name: Aviators**

Name	Roll no	Email
Chaitanya Bramhapurikar	230305	bcvishwas23@iitk.ac.in
Rahul Meena	230832	rahulkcm23@iitk.ac.in
Bhavya Chauhan	230294	bhavyach23@iitk.ac.in
Atharv Phirke	230238	atharvp23@iitk.ac.in
Aaradhya Rohi	230011	aaradhya23@iitk.ac.in
Chaudhari Divyesh	230325	divyesh23@iitk.ac.in
Devansh A Dhok	230354	devanshad23@iitk.ac.in
Areen Mahich	230188	areenm23@iitk.ac.in
Daksh Dua	220321	dakshdua22@iitk.ac.in
Avantika Rohite	230245	avantikar23@iitk.ac.in

Course: CS253

Mentor TA: Souvik Mukherjee

Date: 28 March 2025

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.0	Aviators	Implementation of the software	27/03/25

1 Implementation Details

- **Brief Overview of Frameworks and libraries**

	Back-end	Benefits	Front-end	Benefits
Programming language	Python	**Ease of Development** – Python's simple syntax and readability make backend development faster and more maintainable. **Extensive Library Ecosystem**: With powerful frameworks like Django and Flask, and libraries like SQLAlchemy and NumPy, Python offers robust tools for web development, data processing, and complex backend operations. **Integration Capabilities** – Python seamlessly integrates with databases, APIs, and other technologies. **Cross-Platform Compatibility** – Python runs on multiple platforms, ensuring flexibility in deployment.	HTML	Every browser supports HTML and it is very easy to use.
			CSS	CSS saves time as we can write CSS once and then reuse the same sheet in multiple HTML pages.It is also easy to maintain as to make a global change, simply change the style, and all elements in all the web pages will be updated automatically.
			JS	JavaScript is very fast because it can be run immediately within the client-side browser. Unless outside resources are required, JavaScript is unhindered by network calls to a backend server. Also it is easy to implement.
Frameworks	Django-rest-framework	Why Django-rest-framework? **Fast API development** with built-in tools **Browsable API** for easy testing. **Efficient serialization** and query optimization. **Strong authentication** (OAuth, JWT, etc.)	.	
Libraries	**certifi** – Provides up-to-date CA certificates for secure HTTPS connections. **chardet** – Detects character encodings in text files and web responses. **defusedxml** – Secures XML parsing against vulnerabilities like XML external entity (XXE) attacks. **Django** – High-level Python web framework for rapid development and clean design.		ReactJS	ReactJS allows developers to utilize individual components of an application on both client-side and the server-side. It is also easier to update and manage due to its modular structure.

	django-cors-headers – Handles Cross-Origin Resource Sharing (CORS) in Django applications. **django-filter** – Provides filtering capabilities for Django querysets and REST APIs. **djangorestframework** – Toolkit for building robust REST APIs in Django. **djangorestframework_simplejwt** – Implements JSON Web Token (JWT) authentication for Django REST Framework. **httpie** – Command-line HTTP client for making API requests with a user-friendly interface. **idna** – Supports Internationalized Domain Names (IDN) handling in URLs. **markdown-it-py** – Parses and renders Markdown text in Python. **mdurl** – URL parsing and normalization library for Markdown-related processing. **psycopg2-binary** – PostgreSQL database adapter for Python applications. **PyJWT** – Encodes and decodes JSON Web Tokens (JWT) for authentication. **reportlab** – Generates PDFs in Python for reports and documents. **requests** – Simplifies making HTTP requests in Python applications.. **sqlparse** – Parses and formats SQL queries for better readability. **tzdata** – Timezone database for handling timezone conversions in Python. **urllib3** – Powerful HTTP client for handling connections, pooling, and retries. **whitenoise** – Serves static files efficiently in Django applications.		

DBMS	PostgreSQL	It supports SQL standards, advanced features like JSON storage, full-text search, and MVCC for high performance. It offers strong security, scalability, extensibility with custom functions, and replication for reliability.	NA

Authentication

- **Method**: JSON Web Tokens (JWT)
    - Signing key securely saved on server.
    - *Justification*: Stateless, secure authentication mechanism
    - Benefits:
        - Scalable across different clients
        - No server-side session storage
        - Cryptographically signed

API

- Why REST API?

1. **Decoupled Architecture**
    - Separates frontend and backend concerns
    - Allows independent development and scaling
    - Supports multiple client types (web, mobile, desktop)
2. **Scalability**
    - Stateless communication
    - Easy horizontal scaling
3. **Flexibility**
    - Language-agnostic communication
    - Easy integration with future services
    - Supports multiple data formats (JSON, XML)
4. **Performance**
    - Lightweight data transfer
    - Caching mechanisms
    - Reduced server load compared to traditional server-rendered approaches

Alternative Approaches Considered

- **Monolithic Web Framework (e.g., Django Templates)**
    - Pros: Simpler initial setup
    - Cons: Less flexible, harder to scale, tighter coupling. Harder to separate development

- **GraphQL**
  - Pros: More flexible querying
  - Cons: Higher complexity, steeper learning curve
- **WebSockets**
  - Pros: Real-time communication
  - Cons: More complex implementation, higher resource usage

Chosen Approach: REST API with Django Rest Framework provides the best balance of simplicity, performance, and scalability for a lecture hall booking system where the primary mode of communication is simple HTTPS synchronous requests with JSON data.

API Usage and Implementation Details

1. Authentication

Some API endpoints(`POST/DELETE`) require authentication. Authentication method uses JWT (JSON Web Tokens). Include token in request headers: `Authorization: Bearer <your_token>`

- Endpoint: `/auth/login/`
- Methods:
  - `POST`: user details{ username , password }
  - response : `<your_token>` (valid for 1 hour only)
  -

2. Authorization

Implemented a role based system with four roles:

- Student : Need Approval of authorities to make an approved booking.
- Faculty: Can create a confirmed booking given an available time slot and room.
- Admin : Have access to all CRUD operation on the Database
- Anonymous: Cannot Request booking can only access the live booking schedule and Room details.(`GET` requests)

3. Accounts Management

3.1 User Listing and Creation

- Endpoint: `/accounts/users/`
- Methods:
  - `GET`: List all users (Admin only)
  - `POST`: Create a new user

POST Request Body

{

"username": "string",

"email": "user@example.com",

"password": "string",

"role": "student|faculty|admin",

"authority_email": "optional_email_for_special_roles"

}

3.2 User Management

- Endpoint: `/accounts/users/<user_id>/`
- Methods:
  - `GET`: Retrieve user details (Admin only)
  - `DELETE`: Delete user (Admin only)

4. Bookings Management

4.1 Booking Creation

- Endpoint: `/bookings/new_booking/`
- Method: `POST`

Request Body

{

"title": "Meeting Title",

"room_id": 2,

"start_time": "05:00:00",

"end_time": "07:00:00",

"booking_date": "2025-03-27",

"duration": 2,

"remarks": "Optional additional notes"

“accessories” : [list of accessories]

}

4.2 Booking Operations

- Endpoint: `/bookings/<booking_id>/`
- Methods:
  - `GET`: Retrieve specific booking details
  - `DELETE`: Cancel a booking

GET request example response body


```
[
  {
    "id": 2,
    "room": {
      "id": 1,
      "name": "L-1",
      "capacity": 200,
      "room_type": "lecture_hall",
      "has_ac": false,
      "has_board": false,
      "has_projector": true,
      "price_per_hour": 2000
    },
    "booking_date": "2025-04-26",
    "creator": {
      "id": 12,
      "username": "P_CLUB",
      "email": "pclub@iitk.ac.in",
      "role": "Student",
      "authority_email": [dosa@iitk.ac.in, gensec@iitk.ac.in]
    },
```

```
        "title": "Linux Session",
        "Type": "nonacademic",
        "start_time": "05:00:00",
        "end_time": "06:00:00",
        "requested_on": "2025-03-26T18:58:34.063626+05:30",
        "duration": 1,
        "status": "pending",
        "need_ac": true,
        "need_projector": true,
        "cost": 3000,
        "remarks": for Y-24 batch
    }
]
```


4.3 Booking Search

- Endpoint: `/bookings/search/`
- Method: `GET`

Query Parameters

- `status`: Filter by booking status
- `Type`: Filter by booking type
- `search`: Full-text search across title, room name, creator username
- `ordering`: Sort results

Example: `/bookings/search/?status=approved&Type=non-academic&search=L-1`

4.4 User Booking History

- Endpoint: `/bookings/history/`
- Method: `GET`
- Returns booking history for the authenticated user

5. Room Management

5.1 Room Listing and Creation

- Endpoint: `/bookings/rooms/`
- Methods:
  - `GET`: List all rooms
  - `POST`: Create a new room (Admin only)

GET request example response

{

"id": 1,

"name": "L-1",

"capacity": 200,

"room_type": "lecture_hall",

"has_ac": true,

"has_board": true,

"has_projector": true,

"price_per_hour": 2000

},

5.2 Room Operations

- Endpoint: `/bookings/rooms/<room_id>/`
- Methods:
  - `GET`: Retrieve room details
  - `DELETE`: Delete a room (Admin only)

5.3 Room Search

- Endpoint: `/bookings/rooms/search/`
- Method: `GET`

Query Parameters

- `room_type`
- `has_ac`
- `has_board`
- `has_projector`
- `capacity`
- `price_per_hour`

- `search`: Search by room name

Examples:

- `/bookings/rooms/search/?room_type=lecture_hall&has_ac=true`
- `/bookings/rooms/search/?capacity=200`

6. Reports and Downloads

6.1 Booking Bill Download

- Endpoint: `/reports/download/bill/<booking_id>/`
- Method: `GET`
- Generates PDF bill for a specific booking
- Access restricted to booking creator and admin

6.2 Daily Schedule Download

- Endpoint: `/reports/download/schedule/<date>/`
- Method: `GET`
- Downloads CSV schedule for a specific date
- Date format: `YY-MM-DD`
- Accessible to all authenticated users

7. Approval and Emails

Sent to the concerned authority requesting approval on behalf of the user.
Only applicable to Bookings made by users with ‘Student’ role. Endpoint links can be clicked to perform the operation of approve or reject.

- Endpoint: Approve :`/bookings/approve/<unique-booking-token>`
- Endpoint :Reject :`/bookings/reject/<unique-booking-token>`

8. Error Handling

- 400: Bad Request
- 401: Unauthorized
- 403: Forbidden
- 404: Not Found
- 500: Internal Server Error

9. Rate Limiting : Maximum 100 requests per minute per user.

2 Codebase

GitHub Repository -

https://github.com/chaitanyavb-502/CS253_Automated_Lecture_Hall_Booking_Portal.git

We have built a single repository which contains both the backend as well as the react front-end. We have segregated the code implementation from the documentation into different directories and divided the code implementation into two portions – frontend and backend.

The structure of our frontend is provided on the next page:

Root Directory:

- **.env**: This file likely contains environment variables, such as API keys, base URLs, or other configuration settings that should be kept secret.
- **.gitignore**: Specifies which files and folders Git should ignore (e.g., `node_modules`, `.env`).
- **eslint.config.js**: Configuration file for ESLint, a static code analysis tool used to identify problematic patterns in JavaScript code.
- **index.html**: The main HTML file that serves as the entry point for your React application. It links to the bundled JavaScript files generated by Vite.
- **package-lock.json**: Automatically generated file that tracks the exact versions of dependencies installed.
- **package.json**: Contains metadata about the project, including dependencies, scripts, and project configuration.
- **README.md**: Documentation file explaining how to run, build, and use the project.
- **vite.config.js**: Configuration file for Vite, a build tool that provides fast front-end development experience. It contains settings like server configuration, build options, and plugins.

- **src folder**:Contains the core React components, styles, API handlers, and routes.
    - **App.css**: Contains global CSS styling for your React app.

- **App.jsx**: The main component that serves as the entry point of your React app. It contains routing and the layout structure.
- **frontend_urls.jsx**: A file that contains the frontend URL constants used throughout the application (e.g., `/login`, `/admin`).
- **index.css**: Another CSS file for general styling rules or global resets.
- **main.jsx**: The entry point that renders the `App` component into the DOM.

- **assets folder:** Contains images and static assets used in the application.
  - **assets.js**: An asset manager or an export file for organizing image imports.
  - Various images:
    - **`iitk_logo.png`**:Logo of IIT Kanpur.
    - **`L1.jpg` to `L9.jpg` and `L_18.jpg` to `L_20.jpg`:** Images for different lecture halls or rooms.
    - **`TB1.jpg`** to **`TB5.jpg`**: Images related to tutorial blocks.
    - **`loginpageiitk.png`**: Background image used on the login page.

- **api folder:** Handles backend communication and route protection.
  - api.js: Contains API calls, such as `GET`, `POST`, `PUT`, and `DELETE` requests to interact with the backend.
  - constants.js: Includes constant variables such as API base URLs, endpoints, or reusable keys.
  - protected_route.jsx: Defines protected routes to restrict access to certain pages unless the user is authenticated.

- **components folder:** Contains reusable React components and their styles.
  - **AdminNavbar.jsx & AdminNavbar.css**: Navigation bar component for the admin interface with its styling.
  - **Header.jsx & Header.css**: The header component displayed across pages.
  - **UserNavbar.jsx & UserNavbar.css**: Navigation bar for regular users.
- **pages folder:** Contains React components corresponding to individual pages of the portal. Each `.jsx` file represents a page, and `.css` files contain the styling.

- AdminCreateBooking.jsx & .css: Page for admins to create new bookings.
- AdminHelp.jsx & AdminHelp.css: Help page for the admin.
- Adminviewbooking.jsx & .css: Page for admins to view existing bookings.
- Bookingdetails.jsx & .css: Displays details of a specific booking.
- CreateUser.jsx & .css: Page for creating new users (admin functionality).
- Feedback.jsx: Page for users to submit feedback.
- History.css: Styling for user history pages.
- LiveSchedule.jsx & .css: Displays the current lecture hall booking schedule.
- Login.jsx & .css: Login page for the portal.
- Request_Booking.jsx & .css: Page for users to request a booking.
- RoomDetails.jsx & .css: Displays detailed information about a room.
- Status.jsx & .css: Displays the status of bookings (approved, pending, rejected).
- UserHelp.jsx & UserHelp.css: Help page of user.
- UserHistory.jsx: Displays the history of bookings made by a user.
- Viewpending.jsx & .css: Page to view pending booking requests.

Directory Tree

```
.
├── eslint.config.js
├── fontend.html
├── index.html
├── package.json
├── package-lock.json
├── README.md
├── src
│   ├── api
│   │   ├── api.js
│   │   ├── constants.js
│   │   └── protected_route.jsx
│   ├── App.css
│   ├── App.jsx
│   ├── assets
│   │   ├── assets.js
│   │   ├── elbees_iitk.jpg
│   │   ├── iitk_logo.png
│   │   ├── L_18.jpg
│   │   ├── L_19.jpg
│   │   ├── L1.jpg
│   │   ├── L_20.jpg
│   │   ├── L2.jpg
│   │   ├── L3.jpg
│   │   ├── L4.jpg
│   │   ├── L5.jpg
│   │   ├── L6.jpg
│   │   ├── L7.jpg
│   │   ├── L8.jpg
│   │   ├── L9.jpg
│   │   ├── loginpageiitk.png
│   │   ├── TB1.jpg
│   │   ├── TB2.jpg
│   │   ├── TB3.jpg
│   │   ├── TB4.jpg
│   │   └── TB5.jpg
│   ├── components
│   │   ├── AdminNavbar.css
│   │   ├── AdminNavbar.jsx
│   │   ├── Header.css
│   │   ├── header.jsx
│   │   ├── UserNavbar.css
│   │   └── UserNavbar.jsx
│   ├── frontend_urls.jsx
│   ├── index.css
│   ├── main.jsx
│   └── pages
│       ├── AdminCreateBooking.css
│       ├── AdminCreateBooking.jsx
│       ├── Adminviewbooking.css
│       ├── Adminviewbooking.jsx
│       ├── Bookingdetails.css
│       ├── Bookingdetails.jsx
│       ├── CreateUser.css
│       ├── CreateUser.jsx
│       ├── Feedback.jsx
│       ├── help.jsx
│       ├── History.css
│       ├── LiveSchedule.css
│       ├── LiveSchedule.jsx
│       ├── Login.css
│       ├── login.jsx
│       ├── Request_Booking.css
│       ├── request_booking.jsx
│       ├── RoomDetails.css
│       ├── RoomDetails.jsx
│       ├── Status.jsx
│       ├── UserHistory.jsx
│       ├── Viewpending.css
│       └── Viewpending.jsx
├── tree.html
└── vite.config.js
```

The structure of our backend is :

Root Folder:

- **apps/** : The main part of the codebase, different components of the workings of the backend are divided into modules or apps that work together to provide the required functionality.
- These apps are:
    - **accounts** : User Authentication and Management.
    - **bookings** : Handle room bookings and Automation.
    - **email_services**: Handles email sending.(eg. Booking confirmations)
    - **reports** :Handle report generations in pdf and csv formats for office and admin use.

- The basic structure of each app is:
    - **admin.py** : Registers models to Django Admin
    - **apps.py** : Metadata for Django app
    - **__init__.py** : Marks the directory as a python package
    - **models.py** : Defines database models or tables
    - **serializers.py** : Converts models(database tables) to JSON and validates data for the REST API.
    - **tests.py** : contains unit tests for this app.
    - **urls.py** : Defines URL patterns, API endpoints for this app.
    - **views.py** : Contains API logic as functions and python classes for handling requests. The functions in this file are mapped to specific endpoints defined in the urls.py file.
    - **permissions.py** : Contains custom permission logic to handle access to database and functionalities for different types of user roles(admin, student, faculty).

- **lhbookportal/ :** Main Django project directory, containing project-wide settings, configurations and entry points. Key files:
    - **asgi.py** : ASGI entry point for async communications
    - **__init__.py** : Marks this directory as a Python package.
    - **serializers.py** – Global serializers to apply to the entire project. Currently Used to define a serializer for the JWT authentication system.
    - **settings.py** : Main configuration file (e.g., installed apps, database, middleware, etc.).
    - **urls.py** : Defines the main URL patterns for the entire project. Directs the traffic flow to the respective app based on the url patterns defined here.
    - **views.py** : Global views or views not specific to any app. Used to define views for JWT token authentication.
    - **wsgi.py** : WSGI entry point for running the project on a production server

- **resources/ :** Contains images, pdfs or other static files.

- **server_admin/ :** Admin scripts for automating management of server-side data and operations(eg. adding Holidays, adding fixed Bookings for courses etc.)

- **manage.py** : The main Django CLI tool. Run commands like:
  - *python manage.py runserver* : Start the server
  - *python manage.py migrate* : Apply database migrations(create tables)
  - *python manage.py createsuperuse*r Create an admin user

- **requirements.txt** : Lists required Python packages. Install with:
  - *pip install -r requirements.txt*

- **log.txt** – stores logs for debugging.
- db.sqlite3(Database) : SQLite database storing database tables.
- In total there are 25 directories containing 202 files, among which there are a total of 9 directories containing 51 codebase files which were created by our team.
- Below is the Directory Tree to help with our backend codebase navigation.

Directory Tree

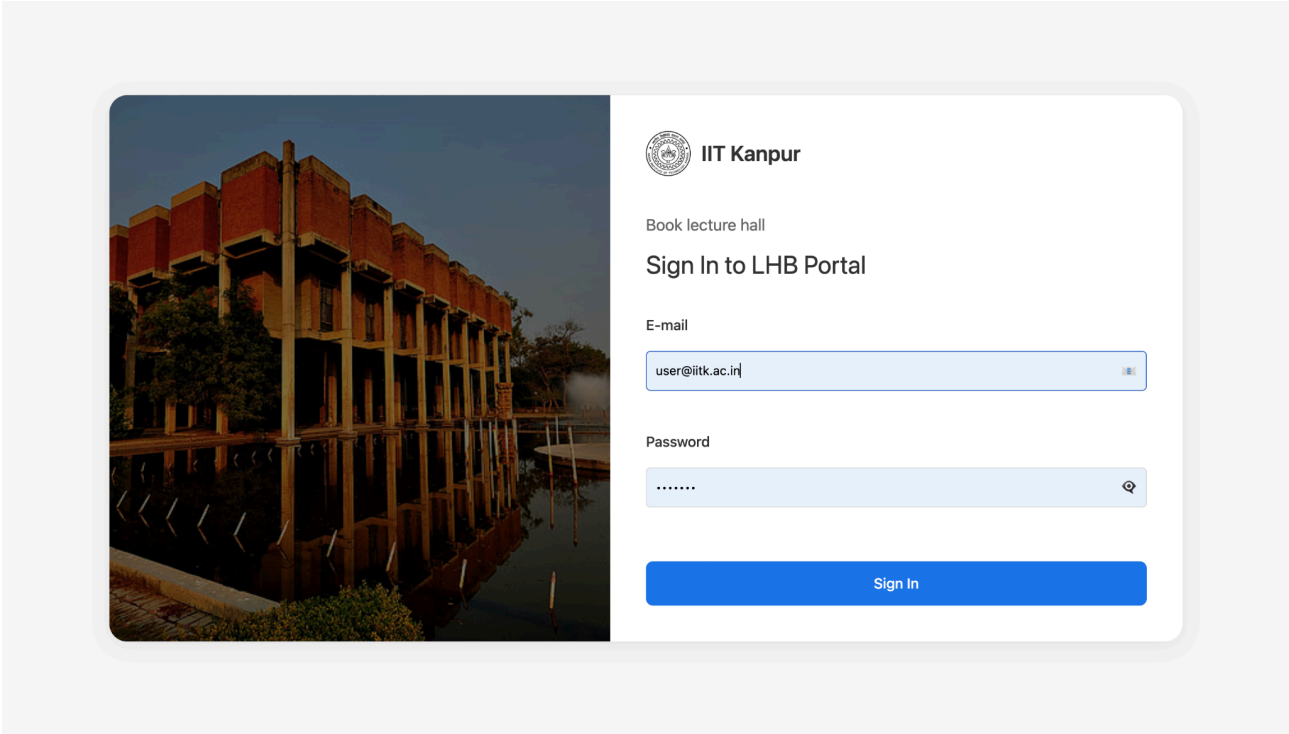
```
.
├── apps
│   ├── accounts
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── __init__.py
│   │   ├── models.py
│   │   ├── serializers.py
│   │   ├── tests.py
│   │   ├── urls.py
│   │   └── views.py
│   ├── bookings
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── __init__.py
│   │   ├── models.py
│   │   ├── permissions.py
│   │   ├── serializers.py
│   │   ├── tests.py
│   │   ├── urls.py
│   │   └── views.py
│   ├── email_services
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── forms.py
│   │   ├── __init__.py
│   │   ├── models.py
│   │   ├── tests.py
│   │   ├── urls.py
│   │   └── views.py
│   └── reports
│       ├── admin.py
│       ├── apps.py
│       ├── images.png
│       ├── __init__.py
│       ├── models.py
│       ├── permissions.py
│       ├── tests.py
│       ├── urls.py
│       └── views.py
├── db.sqlite3
├── lhbookportal
│   ├── asgi.py
│   ├── __init__.py
│   ├── serializers.py
│   ├── settings.py
│   ├── urls.py
│   ├── views.py
│   └── wsgi.py
├── log.txt
├── manage.py
├── project_Tree.txt
├── requirements.txt
├── resources
│   └── images.png
├── server_admin
│   ├── add_holidays.py
│   ├── add_rooms.py
│   └── fixed_bookings.py
```

3 Completeness

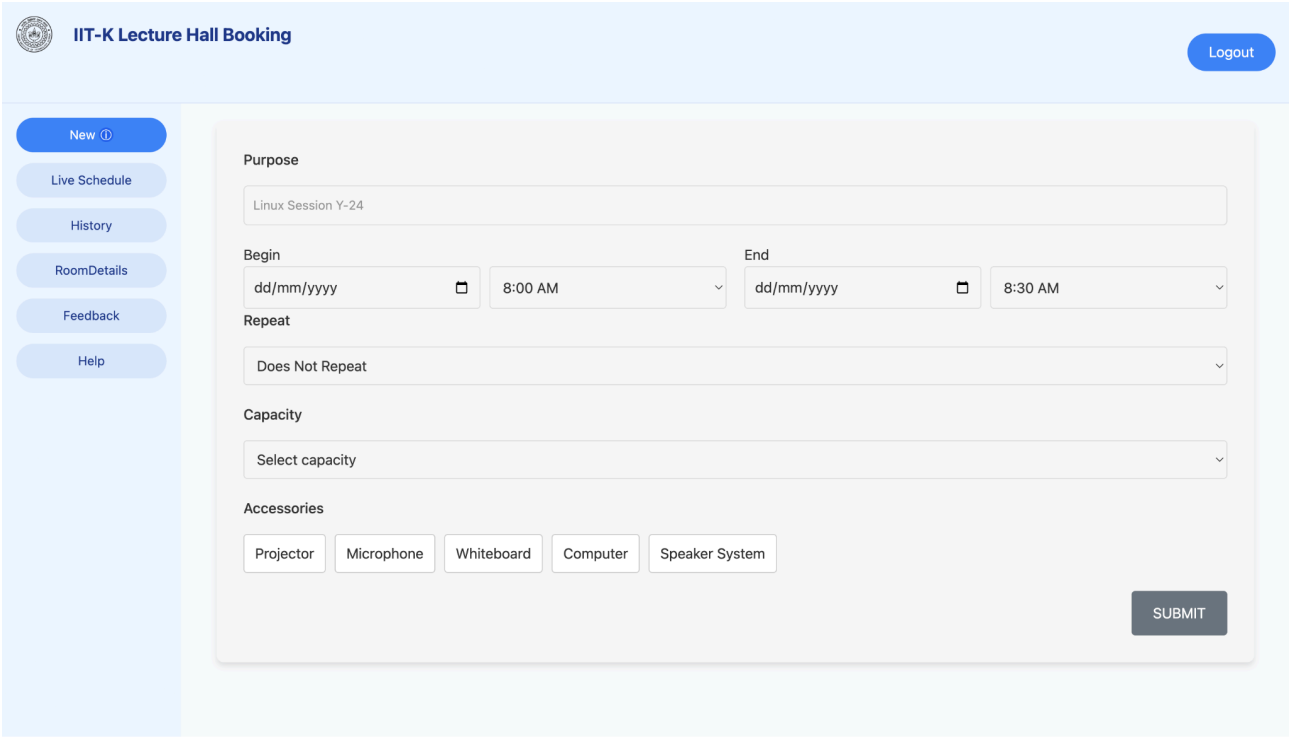
SRS features	Status	Future development plan
User interface	Completed	Providing OTP authentication for security.
Admin interface	Completed	Providing OTP authentication for security. Admin can add new lecture halls.
Timetable display	Completed	NA
Lecture Hall Booking	Completed	Can increase the number of lecture halls. Can make bookings more flexible providing users with available slots.
Email & notification services	Completed	NA
Automated bill generation	Completed	Redirection to payment methods.

USER INTERFACE:

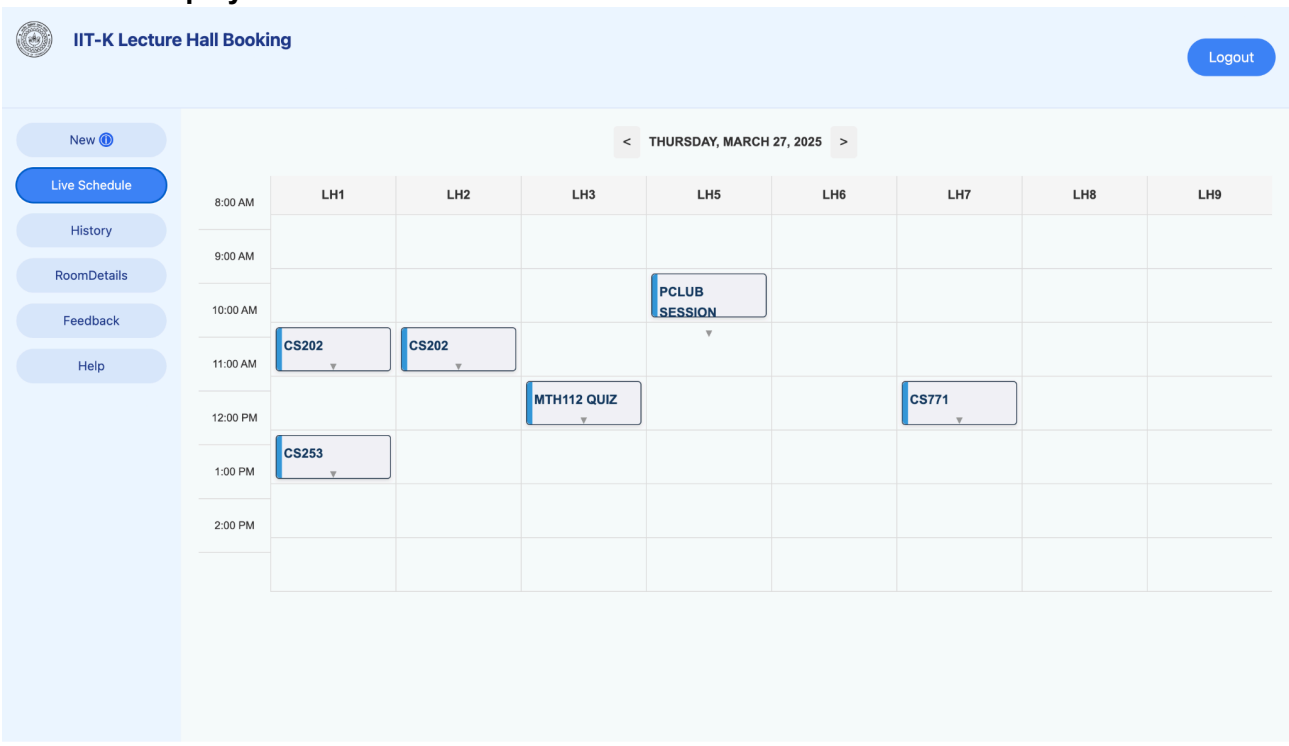
1. **User login page:**



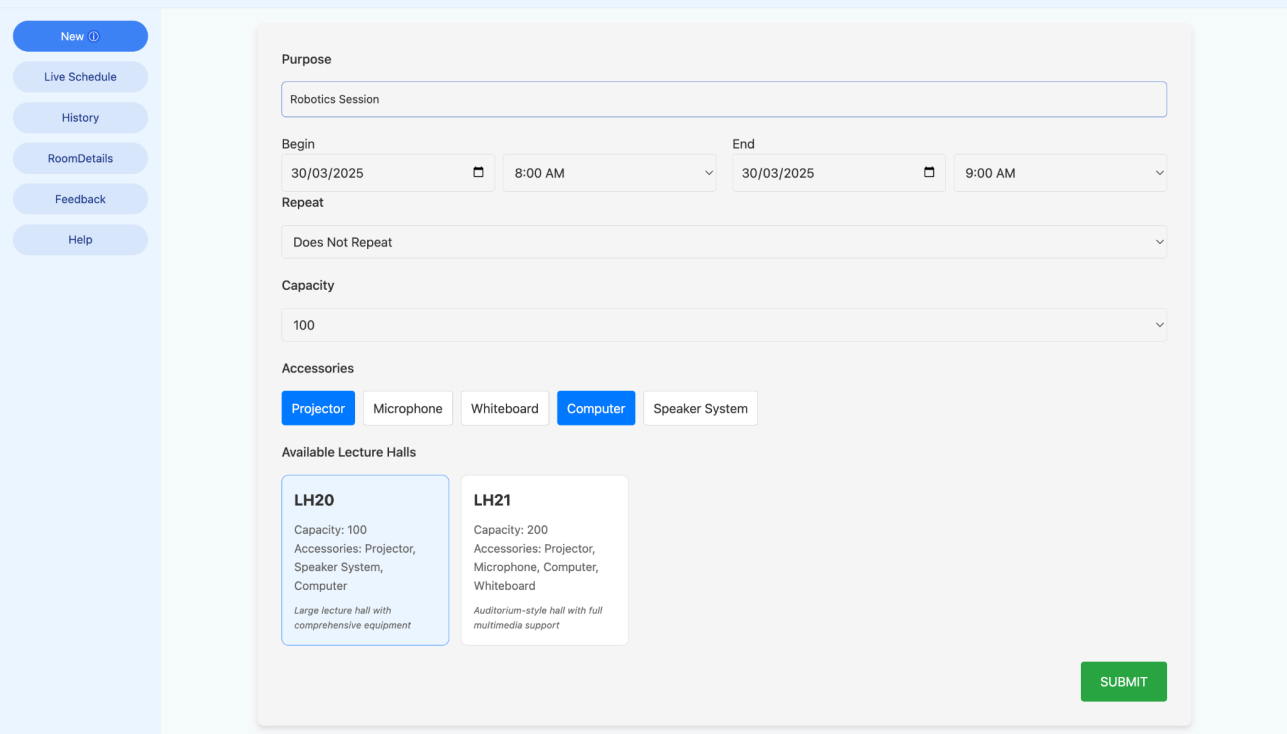
2. User Homepage:



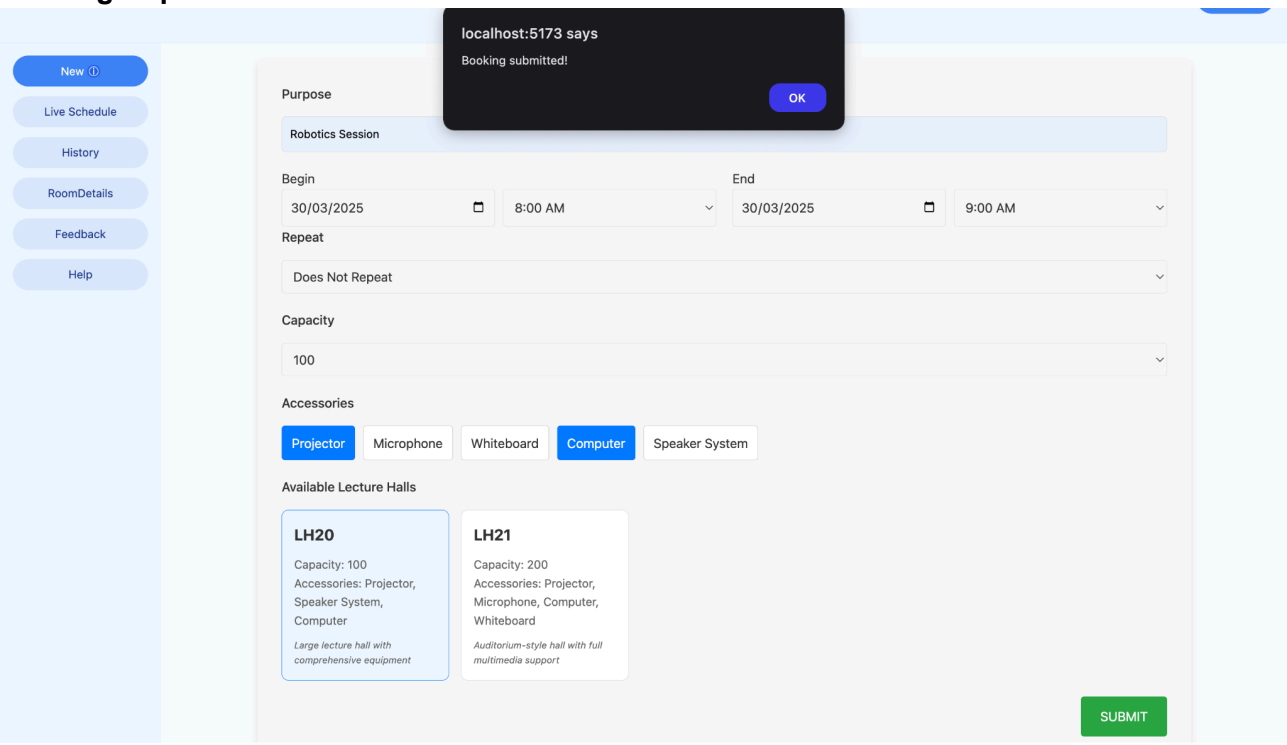
3. Timetable display:



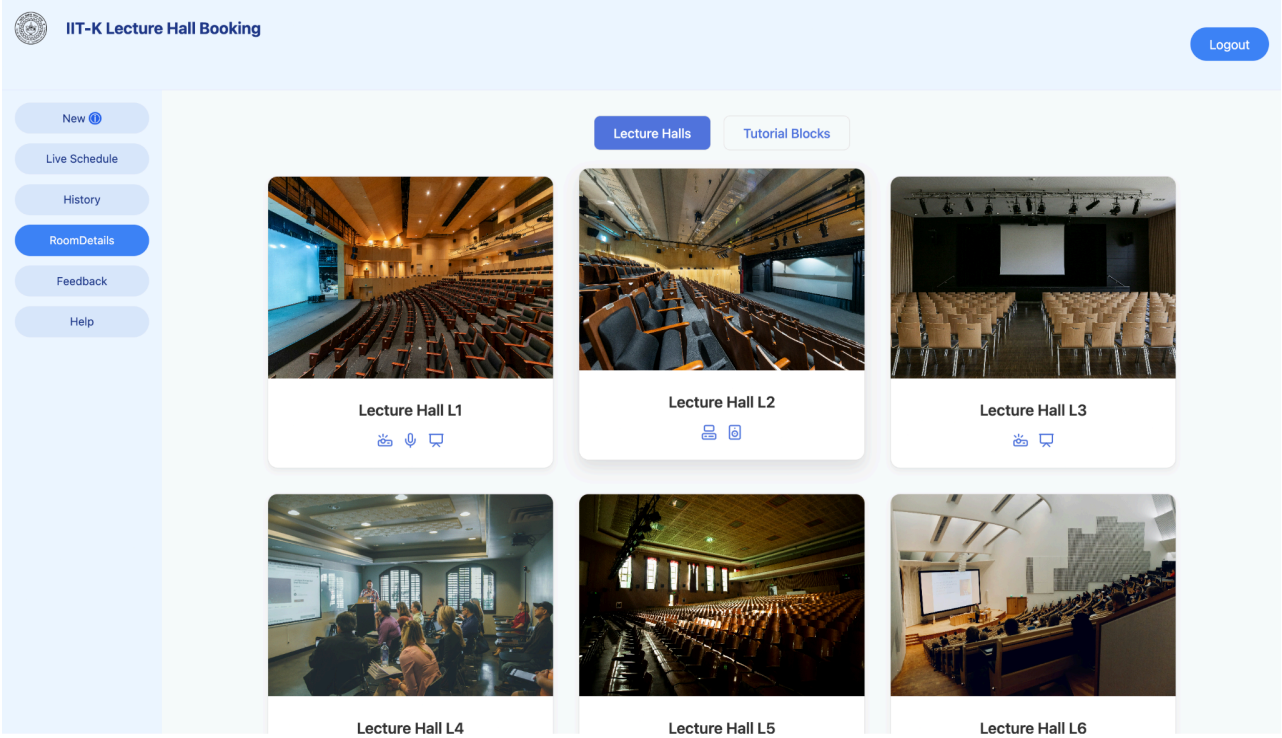
4. Lecture Hall Booking Interface:



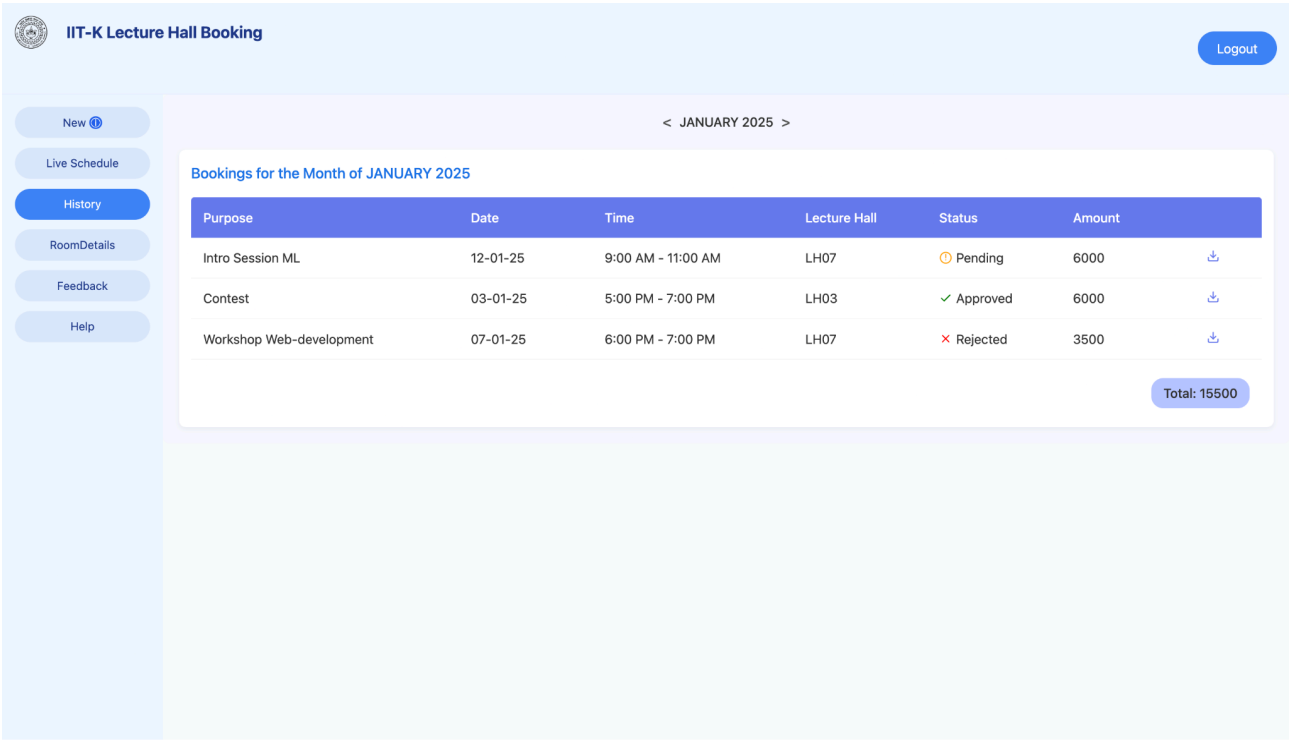
5. Booking request sent:



6. **Room details:** Shows details of lecture halls, such as capacity and accessories that could be provided in that specific lecture hall.



7. **Booking History:** User has access to its booking history.

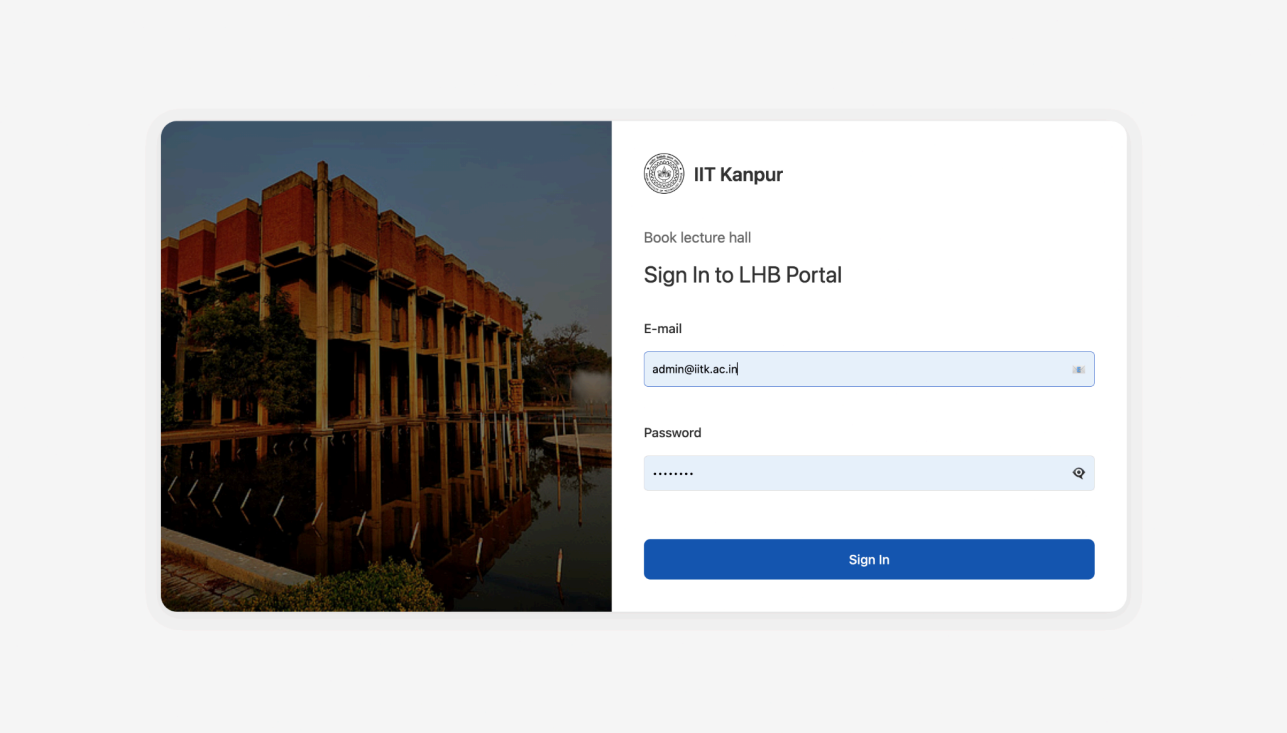


8. **Feedback:** You can give your feedback about the portal and booking system here.

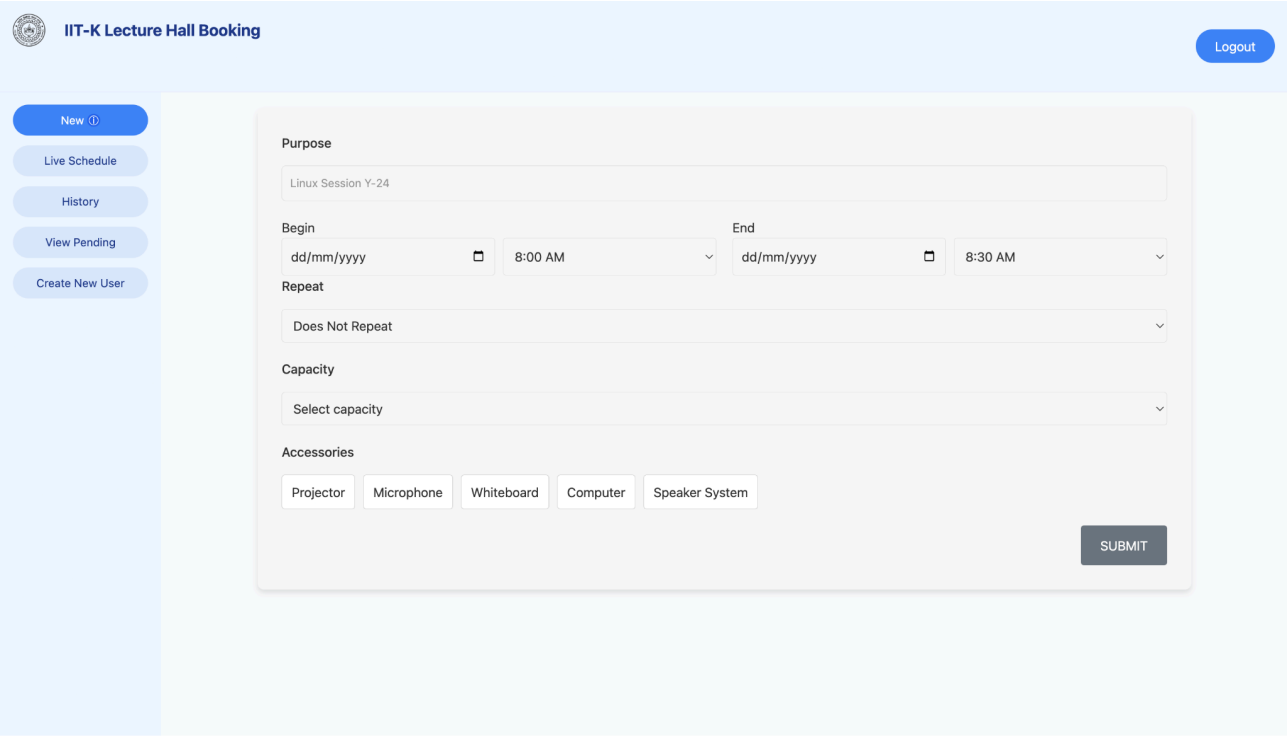
9. **Help :** Guides on how to use the portal.

ADMIN INTERFACE:

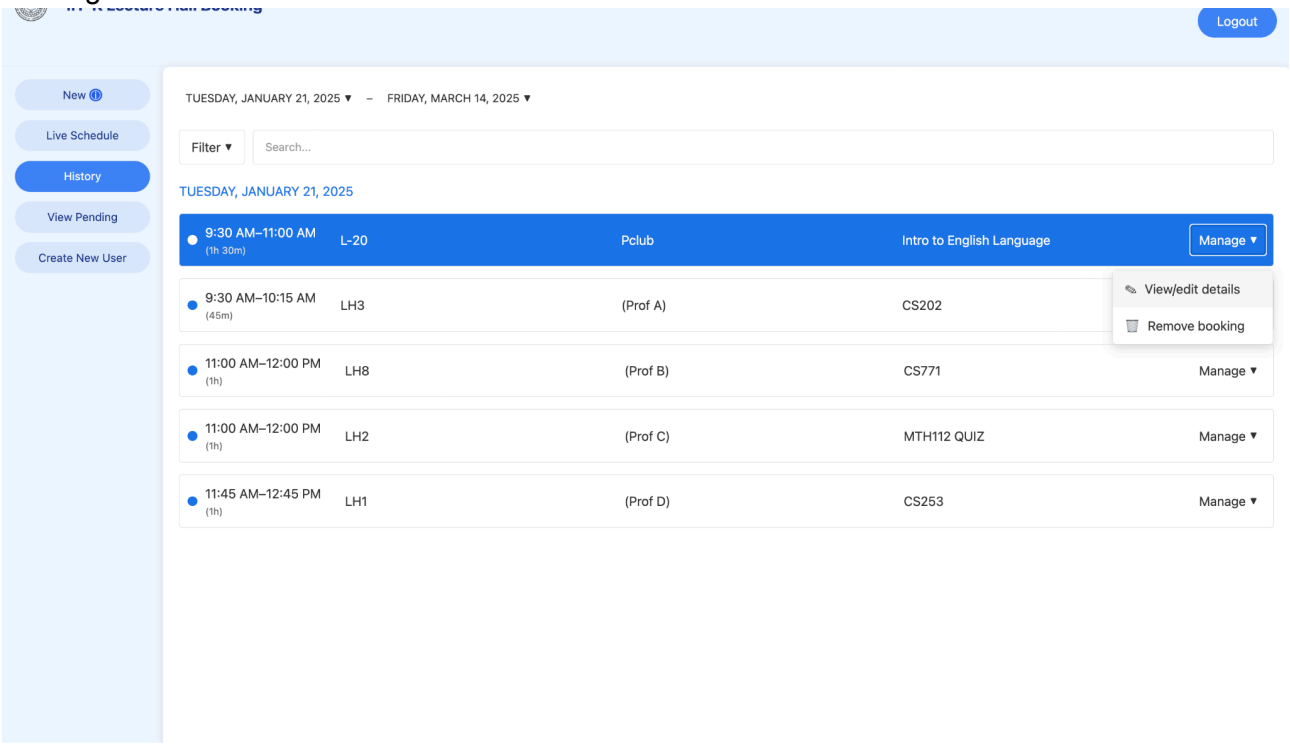
1. **Admin login:** Admin can login here.

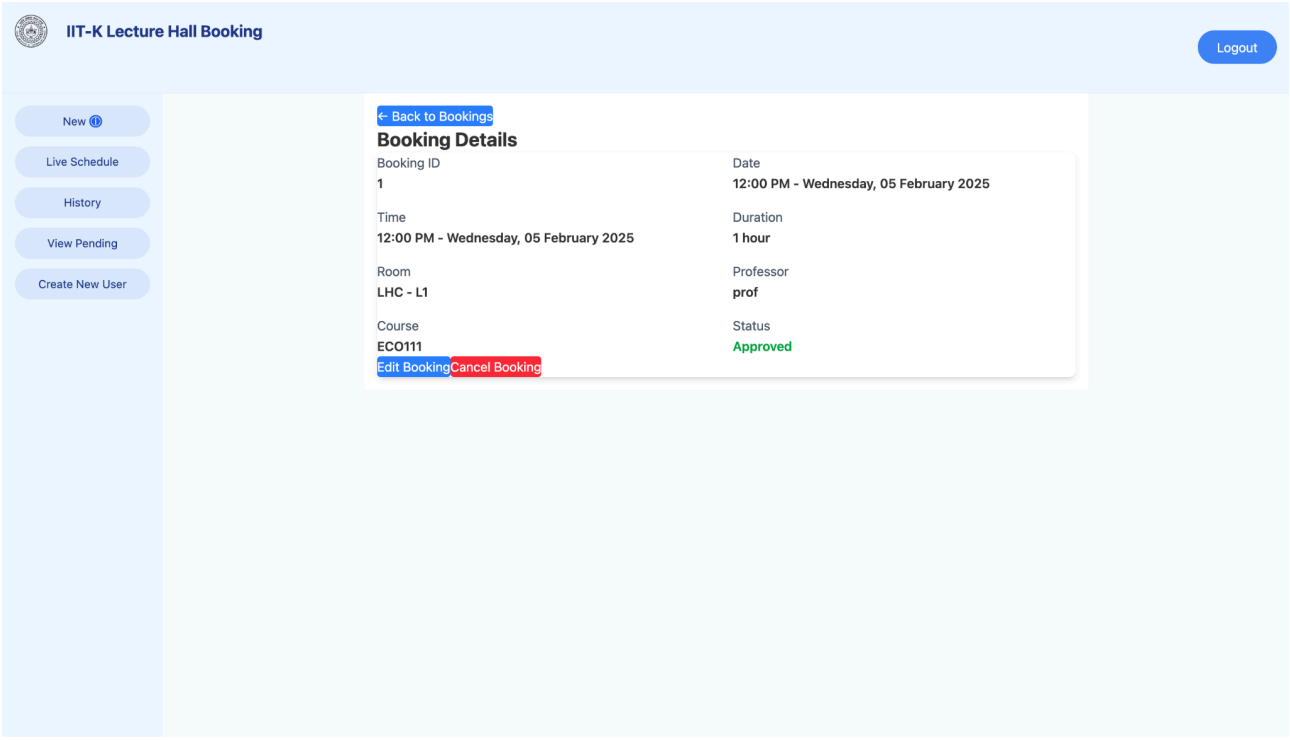


2. **Admin Homepage :**

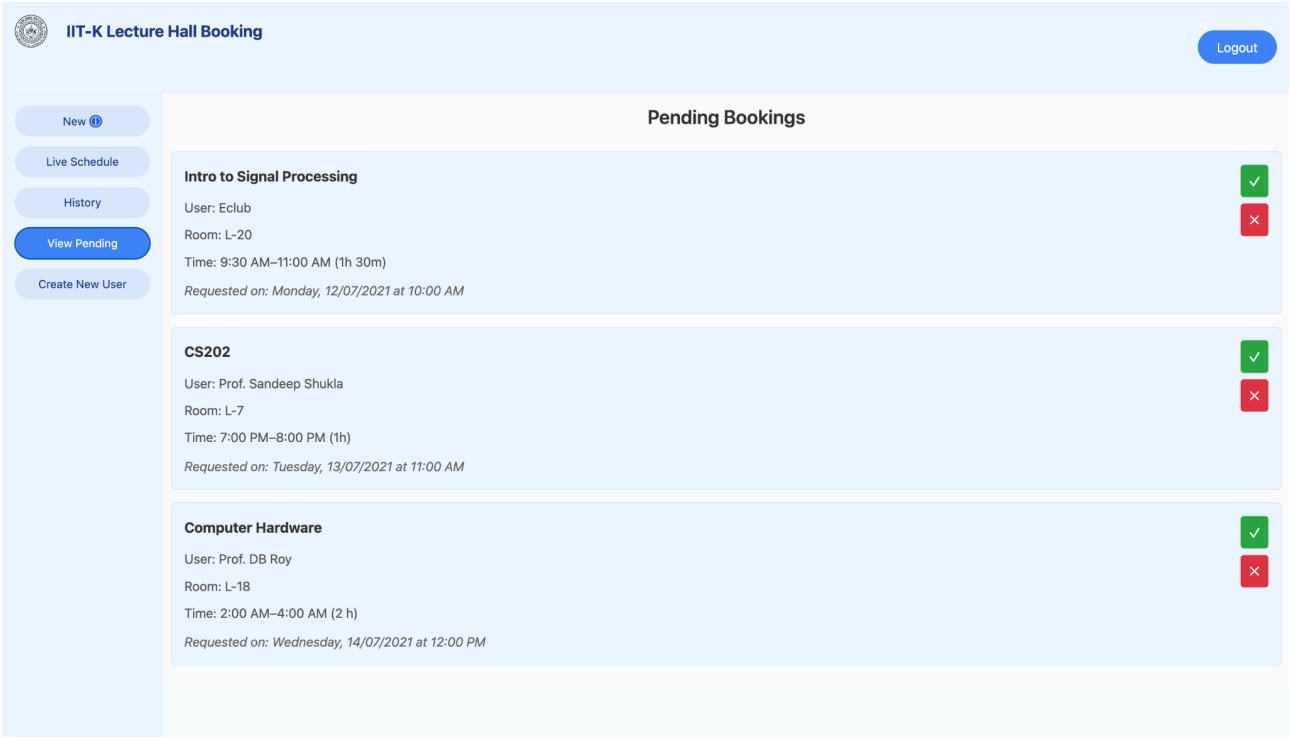


3. **Admin History:** Admin has access to all bookings done and has the freedom to cancel booking.

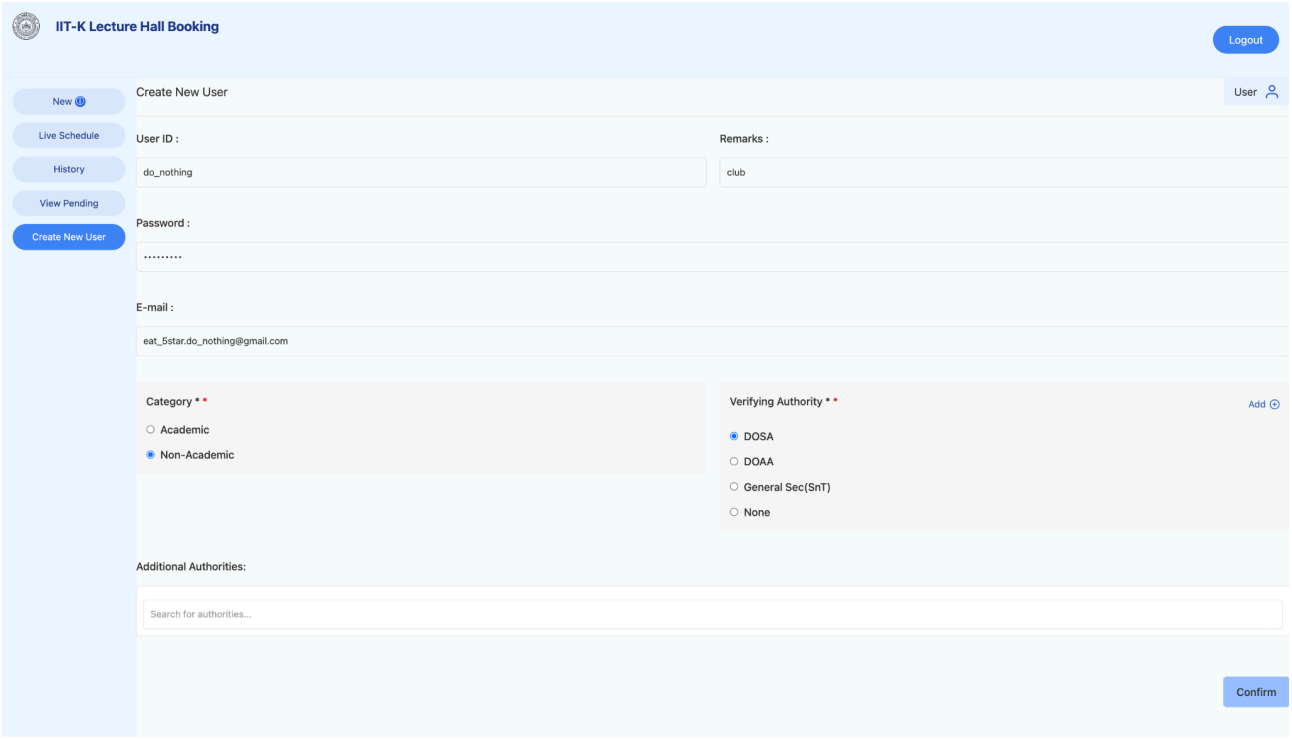




4. View pending bookings:

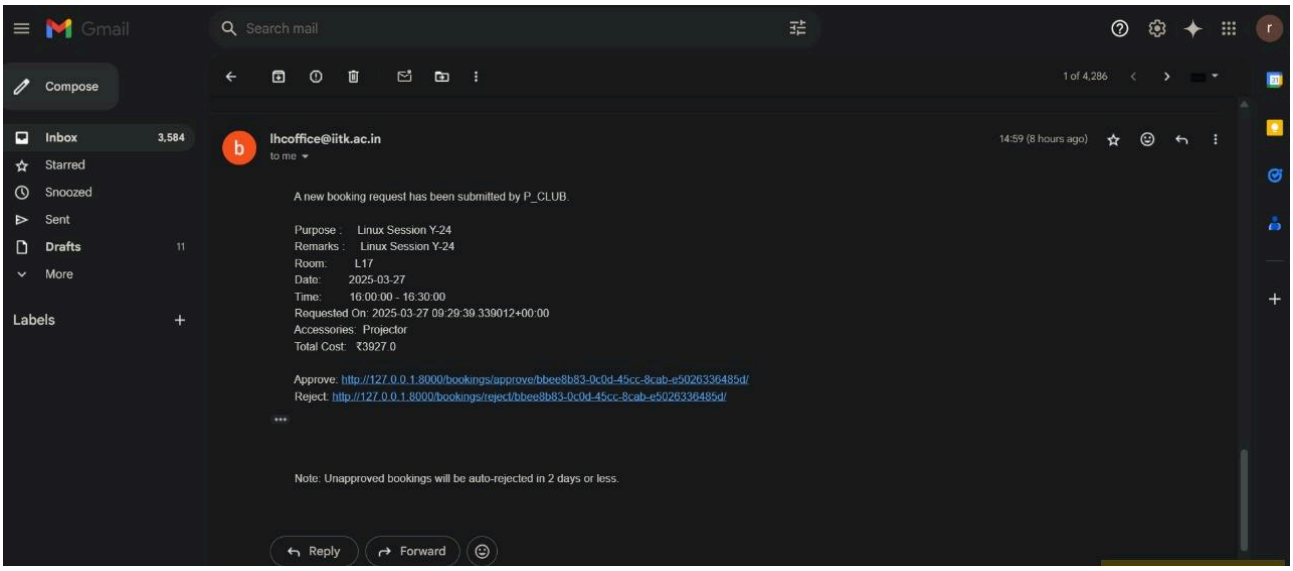


5. Create new user:

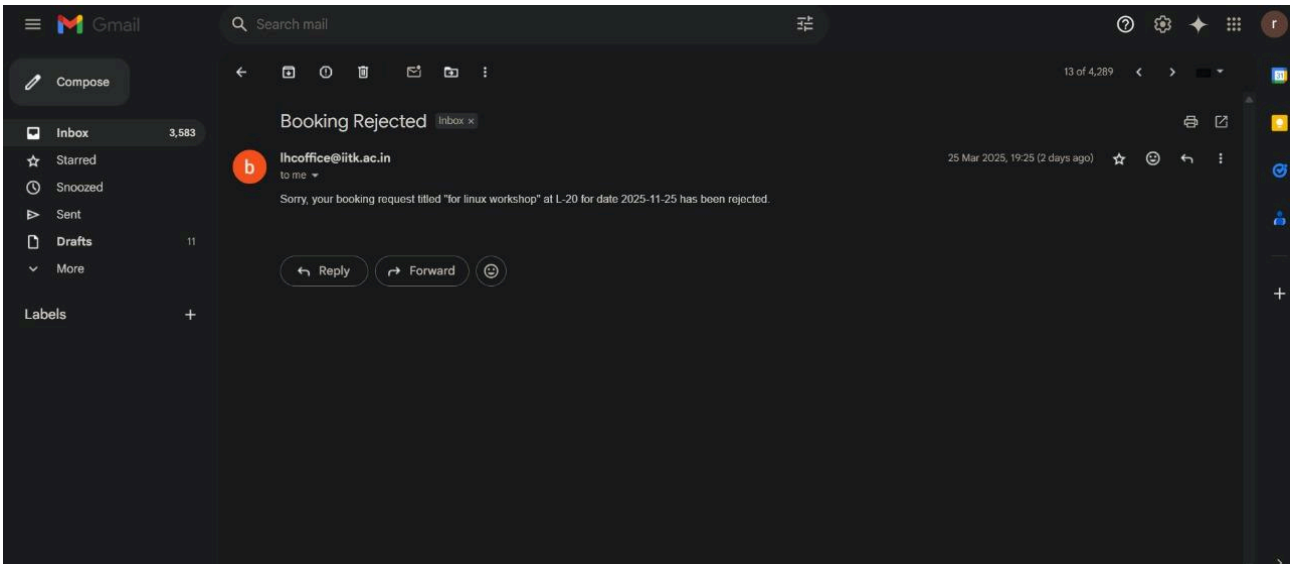


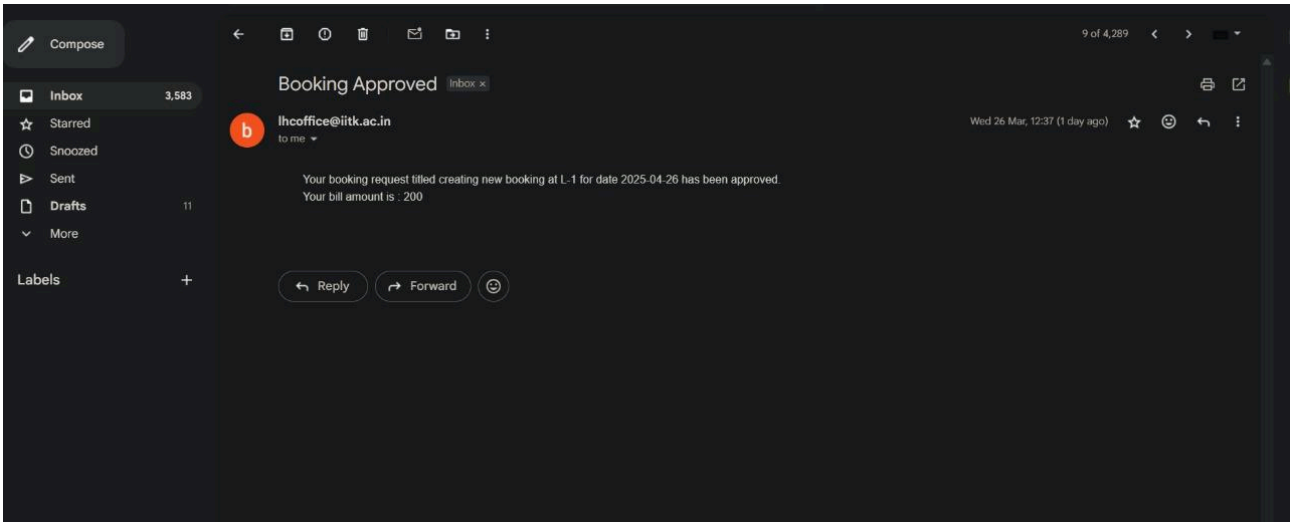
Email/Notification service:

1. **Automated email sent to respective authority:** Authority can approve and reject the request by clicking its respective links provided in the mail.



2. **Notification mail sent to user after action taken by authority:**




Compose
Inbox 3,583
Starred
Snoozed
Sent
Drafts 11
More
Labels
9 of 4,289
Booking Approved
Inbox ×
lhcoffice@iitk.ac.in
to me
Wed 26 Mar, 12:37 (1 day ago)
Your booking request titled creating new booking at L-1 for date 2025-04-26 has been approved.
Your bill amount is : 200
Reply
Forward

Appendix A - Group Log

Team 1 Front-End team	Areen Mahich,Devansh A Dhok,Daksh Dua, Aaradhya
Team 2 Back-End team	Rahul Meena,Bhavya Chauhan,Atharv Phirke, Chaitanya,Avantika

Date	Team	Summary	Duration	Location
27 Feb	All	Divide the team into parts and discuss the timeline	40 min	KD Lab
28 Feb	All	Allocate the respective frontend and backend work among the team	30 min	RM
2 March	1	Discussion regarding the workflow of of the front end	1 hour	RM
3 March	2	Discussion regarding the workflow of email service and authentication permissions .	50 min	Hall 2
5 March	All	Updates regarding the work progress from both teams and considered cross suggestions.	30 min	KD Lab
7 March	1	Discussed the suggestions and made the required changes.	20 min	KD Lab
9 March	2	Tested the email service authentication	45 min	online
10 March	All	Updates regarding the work progress from both teams and considered cross suggestions.	40 min	online
13 March	1	Regarding implementation of frontend codes.	40 min	online
14 March	2	Regarding implementation of backend codes.	1 hours	online

17 March	1	Discussion regarding the changes in the design and the feature of the frontend.	2 hours	CCD
17 March	2	Discussion regarding the changes in some functionalities of Backend.	2 hours	RM building
19 March	All	Discussion regarding the changes made by both teams.	1 hour 30 min	KD Lab
21 March	All	Clarification regarding some changes in the functionalities	2 hours	RM building
26 March	All	Integration of frontend and backend	4 hours	KD Lab
25 March	All	Integration of frontend and backend	3 hours	KD Lab
26 March	All	Clearing bugs and fixing the problems in the system	3 hours 20 min	CCD
27 March	All	Implemented the last adjustments to the documentation.	2 hours	KD Lab